INS FORAT DEL VENT– INTERNATIONAL BACCALUREATE

INFORMÀTICA I - POO

TEMA 5: MÈTODES D'ORDENACIÓ I CERCA

TEMA 5 MÈTODES D'ORDENACIÓ I CERCA

BASE TEÒRICA

- **Concepte**

En computació, l'ordenació i la cerca de dades dins d'una estructura de dades són processos molt utilitzats i de gran rellevància. Disposem de molts algorismes que permeten buscar un element dins d'una col·lecció de dades i/o ordenar una col·lecció de dades en base a algun criteri, per la qual cosa també és important poder mesurar la eficiència de aquests algorismes.

- **Big O**

El Big O és un índex per mesurar l'eficiència d'un algorisme. Concretament intenta mesurar la complexitat computacional (també anomenat cost) i no només la velocitat d'execució d'un algorisme, de manera que mesura el processos que són independents (invariants) del temps dins d'un algorisme, es a dir que és comptabilitza el nombre d'operacions independentment del temps d'execució en base al volum de dades a processar. Així doncs, qualsevol sentència d'assignació, comparació o càlcul, tindrà un temps de execució constant que no dependrà de la mida del conjunt de dades d'entrada, per la qual cosa la seva complexitat serà O (1). Una seqüència de "K" sentències tindrà una complexitat K*O(1). Un bucle de "n" dades serà O(n) i un doble bucle de "n" dades cadascú serà $O(n^2)$.

```
x = 0;                                  → O (1)

x = 0;
y = 10;
x = x + y;                              → O (1) + O(1) + O(1) = 3*O(1) = O(1) (K indep. nombre de dades)

if(x==0)....                            → O (1)  (segons la branca de major complexitat)

for (int i= 0; i < K; i++) { O (1)}  → la complexitat serà K*O(1) = O(1) (K ctn. Indep. de les dades)

for (int i= 0; i < n; i++) { O (1)}  → la complexitat serà n* O (1) = O (n) (n és del nombre dades)

for (int i= 0; i < n; i++) {
        for (int j= 0; j < n; j++) {
                O (1)
        }
}                                       → la complexitat serà n*n* O (1) = O (n²)
```

```
for (int i= 0; i < n; i++) {
        for (int j= 0; j < i; j++) {
                O (1)
        }
}
```
→ la complexitat serà 1 + 2 + 3 + ... + n = n*(1+n)/2 = O(n^2)

```
c= 1;
while (c < n) {
        O (1)
        c= 2*c;
}
```
→ la complexitat serà O (log n) donat que per k d'iteracions c serà 2^k i com que el bucle acabarà per:

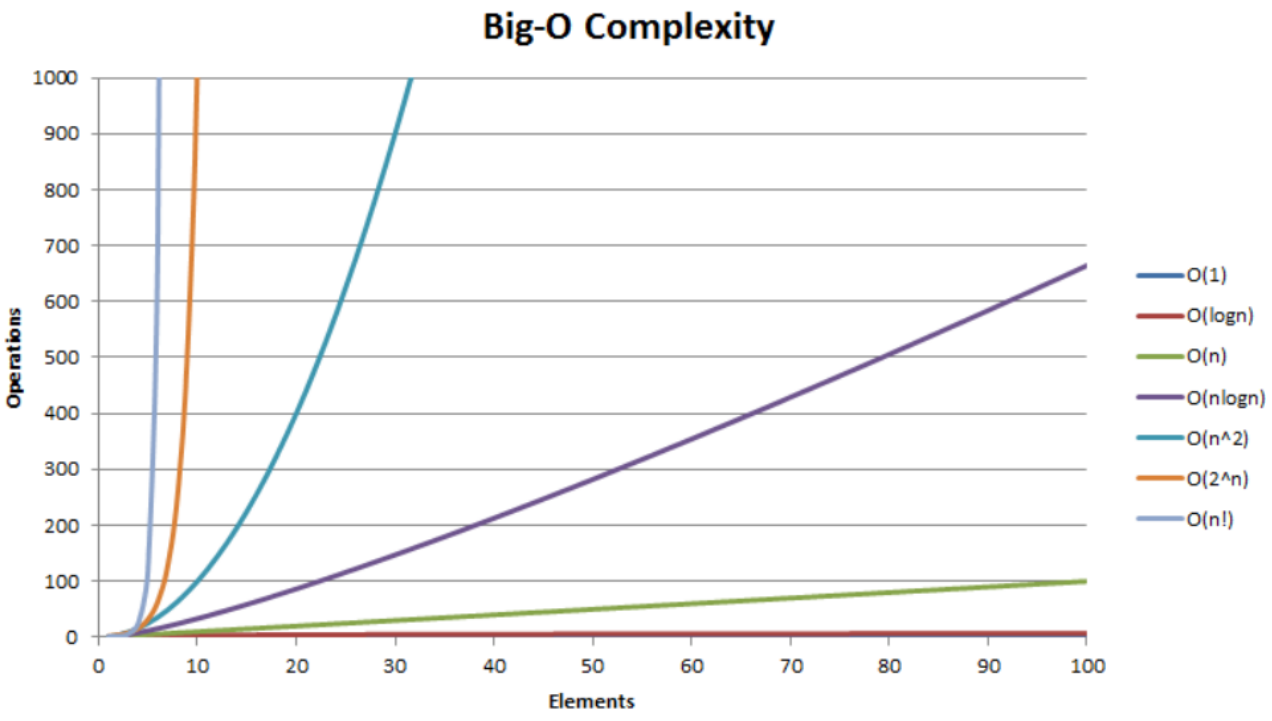
c >= n ➔ 2^k >= n ➔ k= log2 (n)

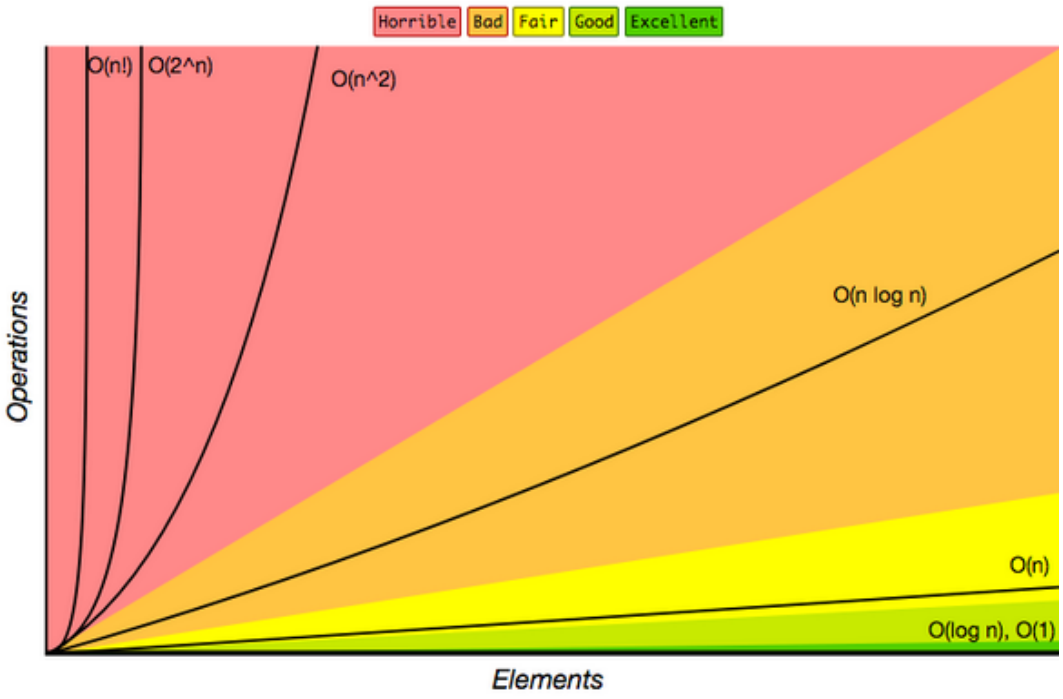
```
for (int i= 0; i < n; i++) {
        c= i;
        while (c > 0) {
                O (1)
                c= c/2;
        }
}
```
→ la complexitat serà O (n*log n), ja que tenim un bucle intern d'ordre O (log n) que s'executa n vegades (bucle extern)

En funció de la mida n del volum de les dades a processar tindrem doncs:

Temps logarísmic (c*log (n))
Temps lineal (c*n),
Temps quadràtic (c*n^2)
Temps polinomial (c*n^k)
Temps exponencial (c^n).

ordenades de menor a major, per a valors de n suficientment grans:



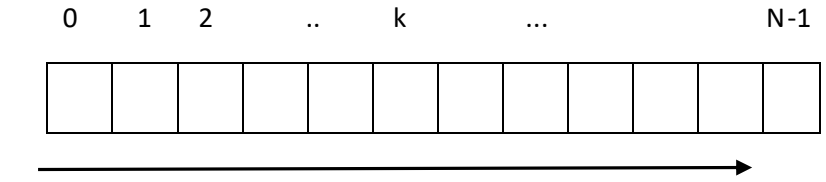


- **Mètodes d'ordenació**

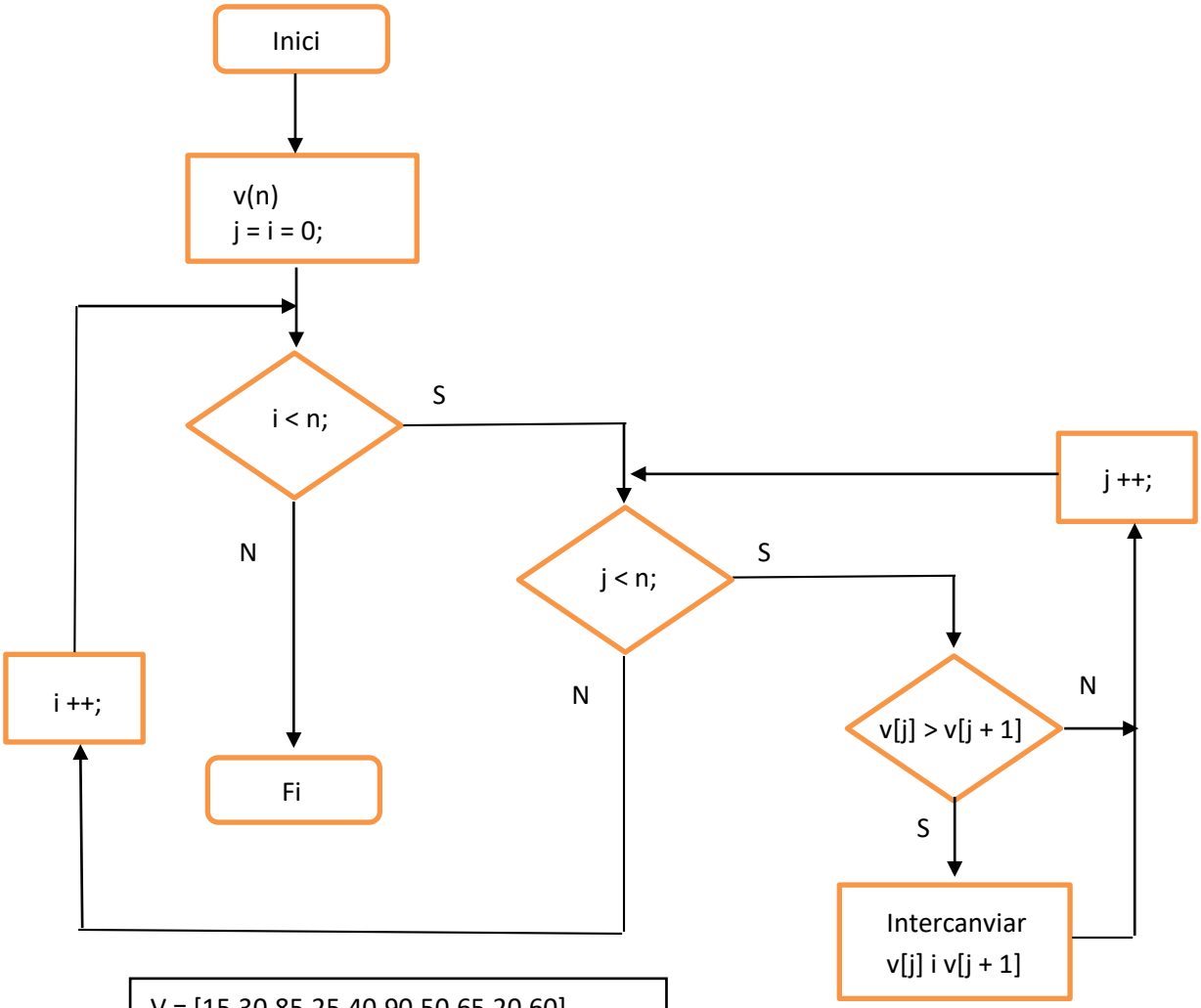
L'ordenació o classificació és el procés d'organitzar un conjunt de dades en algun ordre o seqüència específica, tal com creixent o decreixent per a dades numèriques o l'ordre lexicogràfic per a dades amb caràcters. Els algorismes d'ordenació permuten els elements del conjunt de dades fons aconseguir l'ordre desitjat. Es basen en dues operacions bàsiques: la comparació i l'intercanvi. Hi ha molts algorismes d'ordenació amb diferents avantatges i inconvenients.

Mètode de la bombolla

Consisteix en fer el recorregut del vector ("realització d'una passada") un cert nombre de vegades, comparant parells de valors que ocupen posicions adjacents (0-1,1-2, ...). Si les dues dades no estan ordenades, s'intercanvien. Aquesta operació ES repeteix n-1 vegades, essent n la mida del conjunt de dades d'entrada. Al final de l'última passada l'element major estarà a l'última posició; al final de la segona passada, el segon element quedarà a la penúltima posició, i així successivament. El seu nom es deu a que l'element amb el valor més gran puja a la posició final del vector, de la mateixa manera que les bombolles d'aire en un dipòsit pugen a la part superior.



A cada iteració, el més petit (o gran) és desplaça cap al final del vector




```
V = [15,30,85,25,40,90,50,65,20,60]

output "Before sorting"
loop C from 0 to 9
  output V[C]
end loop

output "========"

loop I from 0 to 8
   loop J from 0 to 8
      if V[J] < V[J + 1] then
        TEMP = V[J]
        V[J] = V[J+1]
        V[J+1] = TEMP
      end if
   end loop
end loop

output "After sorting"
loop C from 0 to 9
  output V[C]
end loop
```

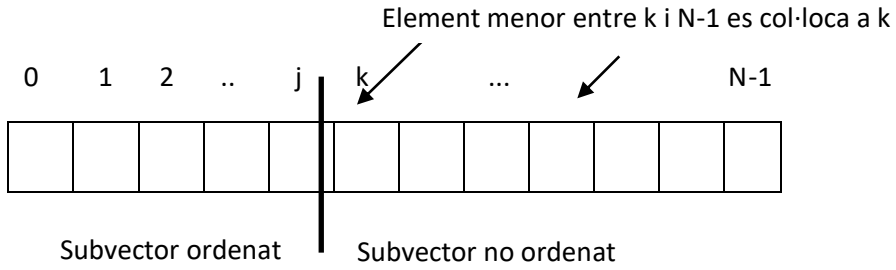
```
package bombolla;
import java.util.Scanner;
public class Bombolla {
   public static void main(String[] args) {
      int n;
      Scanner input = new Scanner(System.in);
      System.out.println("Introduiu la longitud del vector: ");
      n = input.nextInt();
      int v[] = new int[n];
      llegirVector(v);
      System.out.println("SENSE ORDENAR");
      mostrarVector(v);
      System.out.println("ORDENANT SEGONS METODE BONBOLLA");
      ordenaBubble(v);
      System.out.println("ORDENAT");
      mostrarVector(v);
   }
   /* Mètode d'ordenació mitjançant l'algorisme de la bombolla */
   public static void ordenaBubble(int v[]) {
      /* Bucle des de 0 fins la longitud del vector-1 */
      for (int i = 0; i < v.length - 1; i++) {
         for (int j = 0; j < v.length - 1; j++) {
            /* Si el numero a la posició j es major que el de la posició j+1 (el següent del vector)*/
            if (v[j] > v[j + 1]) {
               /* guardem el número de la posició j+1 en una variable auxiliar (el menor) */
               int aux = v[j + 1];
               /* i intercanviem la posició */
               v[j + 1] = v[j];
               v[j] = aux;
               /* i tornem a fer el proces per comparar tots els elements del vector*/
               /* Així anem deixant els números mes grans al final del vector en ordreascendent*/
            }
            mostrarVector(v);
         }
      }
   }
   public static void llegirVector(int v[]) {
      Scanner input = new Scanner(System.in);
      for (int i = 0; i < v.length; i++) {
         System.out.println("Introduiu numero de la posició " + (i + 1) + ": ");
         v[i] = input.nextInt();
      }
   }
   public static void mostrarVector(int v[]) {
      for (int i = 0; i < v.length; i++) {
         if(i == 0)  System.out.print("( " + v[0] + " , ");
         else if (i == v.length-1) System.out.println(v[i] + " )");
         else System.out.print(v[i] + " , ");
      }
   }
}
```

Aquest mètode:

- En general, és un dels mètodes menys eficients. No és pràctic quan el vector és gran.
- Pot ser eficient per a vectors gairebé ordenats.
- La seva complexitat computacional és $O(n^2)$

Mètode de selecció

Aquest mètode considera que el vector està format per dues parts: una part ordenada (la esquerra) que estarà buida al principi i al final comprèn tot el vector; i una part desordenada (la dreta) que al principi comprèn tot el vector i al final serà buida. L'algorisme pren elements de la part dreta i els col·loca a la part esquerra; comença pel menor element de la part desordenada i ho intercanvia amb el que ocupa la seva posició a la part ordenada. Així, a la primera iteració es busca el menor element i s'intercanvia amb què ocupa la posició 0; a la segona, es busca el menor element entre la posició 1 i el final i s'intercanvia amb l'element a la posició 1. D'aquesta manera, les dues primeres posicions del vector estan ordenades i contenen els dos elements menors dins del vector. Aquest procés continua fins a ordenar tots els elements del vector.



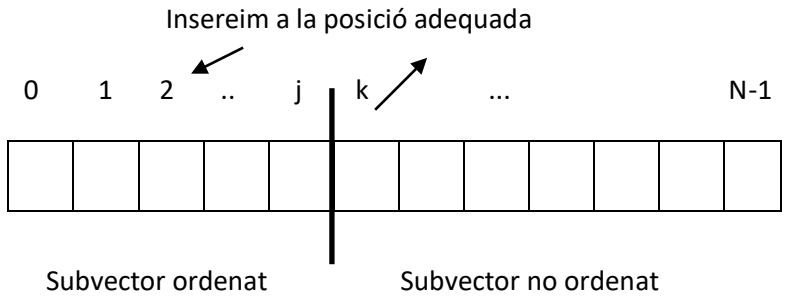

```
//A cada iteració, seleccionem el menor element del subvector
//sense ordenar i s'intercanvia amb el primer element d'aquest subvector.
public static void ordenaSeleccio(int[] v) {
   for (int i = 0; i < v.length - 1; i++) {
      //menor element del subvector v[i..v.length-1]
      int pos_min = i;
      for (int j = i + 1; j < v.length; j++) {
         if (v[j] < v[pos_min]) {
            pos_min = j;
         }
      }
      //Coloca el minim en v[i]
      int aux = v[i];
      v[i] = v[pos_min];
      v[pos_min] = aux;

      mostrarVector(v);
   }
}
```

- Millora una mica el mètode de la bombolla, més si els vector contenen dades complexes
- La seva complexitat computacional és $O(n^2)$

Mètode per inserció

S'utilitza un mètode similar a l'anterior, prenent un element de la part no ordenada per col·locar-lo al seu lloc a la part ordenada. El primer element del vector (v[0]) es considerat ordenat (la llista inicial consta d'un element). A continuació s'insereix el segon element (v[1]) a la posició correcta (davant o darrere de v[0]) depenent que sigui menor o més gran que v[0]. Repetim aquesta operació successivament de tal manera que es va col·locant cada element a la posició correcta. El procés es repetirà N-1 vegades.
Per col·locar la dada al seu lloc, cal trobar la posició que li correspon a la part ordenada i fer-li un forat de manera que es pugui inserir. Per trobar la posició cal trobar un element més gran que el donat. Per fer el buit cal desplaçar els elements pertinents una posició a la dreta.




```
//A cada iteració, s'insereix un element del subnector
//sense ordenar a la posició correcta dins del subvector ordenat.
public static void ordenaInsercio(int[] v) {
   int p, j;
   int aux;
   for (p = 1; p < v.length; p++) {          // des de el segon element fins
       aux = v[p];                           // el final, guardem l'element i
       j = p - 1;                            // comencem a comprovar amb l'anterior
       // mentre quedin posicions i el valor de aux sigui menor que els
       while ((j >= 0) && (aux < v[j])) {
           v[j + 1] = v[j];                  // de l'esquerra, es desplaça a
           j--;                              // la dreta
       }
       v[j + 1] = aux;      // coloquem aux en el seu lloc
   }
}
```

- Millora els mètodes anteriors, especialment quan els vectors estan semiordenats.
- La seva complexitat computacional és $O(n^2)$

Mètode Quicksort: Arrays.sort()

Per ordenar arrays de qualsevol tipus Java disposa del mètode sort de la classe Arrays. Aquest mètode implementa el mètode Quicksort, un dels més eficients. Per utilitzar-lo cal incloure l'import:

```
import java.util.Arrays;
```

Per exemple, donat un array de nom v, de qualsevol tipus de dades, per ordenar-ho de forma ascendent escriurem la instrucció:

```
Arrays.sort(v);
```

Arrays.sort() ordena de forma ascendent (de menor a major). Per ordenar un array de forma descendent (de major a menor) cal indicar-ho utilitzant el mètode reverseOrder() de la classe Collections, caldrà incloure l'import:

```
import java.util.Collections;

Arrays.sort(v, Collections.reverseOrder());
```

També tenim l'opció d'ordenar només una part de l'array, indicant la posició de l'element inicial i la de l'element final (que no s'inclou a l'ordenació).

```
Arrays.sort(v, 1, 4);
```

L'1 indica la posició de l'element on comença l'ordenació i el 4 indica la posició del primer element que no entra a l'ordenació.

```
import java.util.Arrays;
import java.util.Scanner;

public class Ordenarcio {
   public static void main(String[] args) {
      //int [] v = {3,7,1,0,-2,2,5,-1};
      int n;
      Scanner input = new Scanner(System.in);
      System.out.println("Introduiu la longitud del vector: ");
      n = input.nextInt();
      int v[] = new int[n];
      llegirVector(v);
      System.out.println("SENSE ORDENAR");
      mostrarVector(v);
      System.out.println("ORDENANT SEGONS METODE BONBOLLA");
      Arrays.sort(v);
      System.out.println("ORDENAT");
      mostrarVector(v);
   }
```

- **Mètodes de cerca**

Cerca lineal o seqüèncial

Consisteix en anar comparant els elements del vector un a un fins trobar la posició en que la dada coincideix amb l'element buscat. En aquest cas es proporciona la posició del vector en la qual s'ha trobat l'element buscat. En cas que no es trobi l'element en qüestió, es tornarà un -1.

```
import java.util.Scanner;

public class Busquedalin {
   public static void main(String[] args) {
      int [] dades = {9,3,5,-1,0,9,5,11};
      Scanner input = new Scanner(System.in);
      System.out.println("Introuduiu el numero a buscar en el vector:");
      int n = input.nextInt();
      int pos = bus_lin(dades, n);
      if(pos!=-1) System.out.println("El numero esta al vector a la posicio " + pos);
      else System.out.println("El numero NO esta al vector");
   }

   public static int bus_lin(int [] dades, int n) {
          int pos = -1;
          for ( int i = 0 ; i < dades.length; i++) {
                  if (n == dades[i]) pos = i;
          }
          return pos;
   }
}
```

La seva complexitat és de l'ordre de : O(n).

Cerca binaria

Per poder aplicar aquest mètode, es condició que el vector estigui ordenat (Pre-condició). El mètode consisteix en comparar l'element buscat amb l'element situat al mig (centre) del vector:

- Si coincideixen, s'ha trobat l'element buscat.
- Si l'element buscat és més gran que l'element central del vector, haurem de continuar la cerca sobre la meitat superior del vector.
- Si l'element buscat és més petit que l'element central del vector, haurem de continuar la cerca sobre la meitat inferior del vector

La seva complexitat és de l'ordre de : O(log n).

```
public static int bus_bin (int [] dades, int n ) {
  int max = dades.length -1;
  int min = 0;
  while (max >= min) {
      int middle = (min+max)/2;
      if (dades[middle] == n) return middle;
      else if (dades[middle] > n)  max = middle - 1 ;
      else if (dades[middle] < n)  min = middle + 1 ;
      else return (-1);
  }
   return (-1);
}
```

- **Programes per desenvolupar**

1.- Fer els diagrames de flux i el pseudocodi corresponent als mètodes d'ordenació de selecció i inserció i al mètodes de cerca lineal i binari.

2.- Proposa alguna millora per qualsevol dels mètodes presentats.

- **Programes opcionals**

1.- Explica en què consisteix el mètode Quicksort.